0655/63676

AUSTRALIA

**Patent Office**
**Canberra**

# CERTIFIED COPY OF PRIORITY DOCUMENT

I, LISA TREVERROW, TEAM LEADER EXAMINATION SUPPORT AND SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PQ 4284 for a patent by COMPUTER ASSOCIATES PTY LTD. filed on 26 November 1999.

WITNESS my hand this
Twelfth day of December 2000

Lisa Treverrow

LISA TREVERROW
TEAM LEADER EXAMINATION
SUPPORT AND SALES

P/00/009 28/5/91
Regulation 3.2

AUSTRALIA

*Patents Act 1990*

# ORIGINAL

## PROVISIONAL SPECIFICATION

**A METHOD AND APPARATUS FOR OPERATING A DATA BASE**

**The invention is described in the following statement:**

# A METHOD AND APPARATUS FOR OPERATING A DATABASE.

## FIELD OF INVENTION

The present invention relates to databases, particularly relational databases and RDBMS (Relational Database Management System) implementations. The present invention also relates to databases or systems used to directory services, such as X.500 service directories and LDAP service directories.

In particular, the present invention relates to the operation of a database and / or it's application(s). An example use of the present invention, but not its exclusive use, is in respect of the directory services database disclosed in PCT/AU95/00560. The disclosure relates to a relational database for performing X.500 and LDAP services and discloses methods of operation, therein, in particular relational databases that support SQL.

One, but not the only, aspect of the present invention is in respect of search services.

## BACKGROUND OF INVENTION

There is considered to be a performance problem with the running of applications on prior art databases, such as those used for service directories where it is not uncommon to have hundreds of thousands, or even millions, of entries in the database. Users seem to be constantly seeking ever increasing performance and accuracy. The performance of these large databases is often considered to be slow (from a user's perspective), in that, if a search is performed on such a large database, the search may take 1 or 2 seconds to provide a result. In fact, really complex searches may take minutes or even hours to execute. From a user's perspective, this sort of delay is considered irritating. User's prefer to have little, if any, delay in operating a database system.

This perception of lack of performance or lack of speed is especially noticeable when executing a search instruction/service. In essence, the problem is that in executing a search instruction/ service, the application has to take a complex directory query, such as a complex X.500 query, and map that query into SQL. The SQL performed is very much based on a table design, and these tables may use metadata. Even though the step of mapping the query is possible, conventionally the mapping is complex or produces relatively complex SQL expressions to be executed by the database application. Nonetheless, users want the instruction/service executed in such a way that the system exhibits higher performance.

Part of what aggravates the performance problem is that, in a metadata design such as that exemplified in the database disclosed in PCT/AU95/00560, the tables tend to be very large tables. The larger a table grows, the worse may be the application's perceived performance. Also, if the tables are large and a complex query joins a number of the large tables the perceived performance problem gets worse.

To illustrate this problem, an example will be described of the mapping of a relatively complex X.500 query into SQL. The problem with the mapping is that X.500 and LDAP basically gives a query language that involves AND's, OR's and NOT's. SQL also gives a query language which involves AND's, OR's and NOT's. A complex SQL expression may use one or more joins and / or subselects. When there are a number of joins and / or subselects, the query may take a relatively long time to evaluate. There may be a number of ways to write an SQL query and if it only contains joins and not subselects then it is often termed a "flattened" query.

The illustration will be made in respect of a relatively simple example. Looking at Figure 1, a search for an X.500 query using LDAP notation is expressed in equation 101. The search example uses a filter to look for a common name = Rick, AND surname = Harvey. The figure shows a search table 102 and also an attribute table 103 for reference. Lets suppose that attribute identifier AID number 3 is common name (cn) and AID number 4 is surname (sn). In the search table there are a number of AIDs. The Rick Harvey entry is EID entry 101 with AID 3, corresponding to a common name 'RICK' and also EID entry 101 has 'HARVEY' for the normalised value. There is also another AID 3 in the table for John Smith, John who has EID 200 and Smith who has EID 200. In doing a search for Rick AND Harvey, the search will try to find all objects (EIDs) who have AID = 3, norm = RICK, and AID = 4, norm = HARVEY.

So, in essence, the search wants to select entries 'Rick' and 'Harvey'.

One way of doing this search is by using a subselect ( or nested query). The SQL required for this is:

```
Select EID from search
Where AID=3 AND NORM='RICK'
AND EID IN
        (select EID from SEARCH
                where AID=4 AND NORM = 'HARVEY')
```

In this nested query, the clause in brackets is a subselect. The subselect is evaluated corresponding to where AID = 4 and Norm = HARVEY and the resulting list of EIDs is saved. Then the outer clause is evaluated corresponding to where AID = 4 and Norm = HARVEY such that the list of EIDs returned is also in the list of EIDs previously saved.

The problem with the sub-select is that if there are many, many 'HARVEY's a huge set will be built and there may not be many 'RICK's and thus this query will be lopsided and may take a long time to evaluate.

Another way of doing this search is by using a join (or flattened query). The SQL required for this join is:

```
Select S1.EID from search S1, search S2
        Where S1.AID=3 AND S1.NORM = 'RICK'
        AND S2.AID=4 AND S2.NORM = 'HARVEY'
        AND S1.EID = S2.EID
```

The result is that if table S1 has a million entries, and table S2 has a million entries, the search may be conducted over a huge combination of entries when the tables are joined. As can be seen, even for this quite simple search/ instruction, performance of an application can be severely diminished in a relatively large database. However, usually a join version of a query will be faster than a subselect.

The sub-select is equivalent to the join, and in fact many prior art database applications may convert a sub-select into a join. However, this may be too difficult if there is more than one level of nesting or the where clause is too complex.

A further example will now be discussed in which we discuss a search involving a 'NOT' instruction. In other words, in this example a filter asks for 'not equal'. We are looking for 'RICK' NOT 'HARVEY' that is common name is Rick and surname not equals Harvey.

The nested query, in SQL would be:

```
Select EID from search
Where AID =3 AND NORM = 'RICK'
AND EID NOT IN
        (select EID from search
        where AID =4 AND NORM = 'HARVEY')
```

The flatten version of the above query may be accomplished with an outer join. An outer join is considered to be quite complex and relatively slow in execution. An outer join for this search would be something like:

```
SELECT S1.EID FROM
        (SEARCH S1 LEFT JOIN SEARCH S2
                ON S1.EID = S2.EID
                AND S2.AID=4)
Where S1.AID=3
        AND S1.NORM = 'RICK'
        AND (S2.NORM< > 'HARVEY'
                OR S2.NORM IS NULL)
```

The above example relates to a relatively simple search for 'Rick' and not 'Harvey'; in other words, a search of sets involving set 'A' and not set 'B' i.e $A.\overline{B}$

If we look at a search involving more complex searches, such as

$A.(B+C.\overline{D})$, ...................eqn 201

the SQL for this query would be (in abstract):

```
SELECT 'A'
        AND EID IN
                (SELECT 'B' OR
                        (SELECT 'C' AND EID NOT IN
                                (SELECT 'D')))
```

Note that the above query is very difficult to flatten into an expression that involves only joins.

It has been found that the more complex the SQL query, the slower the application will run. Furthermore, it has been found that generally simple 'joins' run faster than 'nested' queries. However, generally outer joins run slower than nested queries especially if they are cascaded. It has also been difficult to find a generalised method of dealing with the problems noted above.

An object of the invention is to alleviate at least one problem of the prior art.

A further object of the present invention is to, where possible, improve the performance of in queries applications.

SUMMARY OF INVENTION

The present invention provides, in one aspect, a method of operating a database in order to execute a service query, the method including the steps of:

receiving a service query,

applying De Morgon's theorem to the service query to obtain a sum of min terms,

evaluate each min terms as one or more separate SQL instructions,

execute each separate SQL instruction.

Preferably, the sum of min terms are additionally expanded to remove 'NOT's, for example by application of Boolean logic.

In a further aspect, the present invention provides, in executing a separate SQL instruction, listing results of subtracted or negative min terms in a 'NOT EID' list, listing results of non-subtracted or additive min terms in an 'EID' list, that is not in the 'NOT EID' list, and not listing or deleting duplicate results or results already listed.

In principle, the present invention seeks to simplify complex queries by reducing the query to a set of smaller or simpler SQL's. The invention has come about because of the realisation that the prior art performance issues can in part be addressed by decompiling or expanding a search instruction into min terms by the application of De Morgan's theorem, anaylising the min terms to remove NOTs and expanding out the result. A further benefit is obtained by disregarding duplicate terms. Each min term can be evaluated as a separate SQL. In one form, a min term may be an ANDed set of items.

The present invention, in experimental testing, has shown significant performance improvements when compared to prior art systems. It is known that for relatively complex queries, execution time increases almost exponentially with increases in complexity of the query. The present invention, in seeking to simplify the complexity, reduces this exponential increase in execution time.

As noted above, the present invention reduces complex queries to a set of min terms. The present invention further seeks to evaluate the set of min terms in order to determine whether there is a relatively more efficient method of execution of the min terms. Furthermore, size and time limits of the instructions are improved in accordance with the present invention.

A preferred embodiment of the present invention will now be described, with reference to the accompanying drawings, in which:

Figure 1 illustrates a prior art example of a search for “Rick’ and ‘Harvey’, and

Figure 2 illustrates graphically an equation used in explaining the operation of the present invention.

The present invention in its various aspects includes a number of steps. In the following description we will provide an illustrative embodiment of those steps.

DETERMINING AN EQUATION

In essence, a service query can be represented mathematically as an equation. It is not necessary to represent a service query as an equation, but for the purposes of illustrating the present invention referring to equations does help to explain the present invention.

If we assume a service query, such as a search query, can be represented by a relatively complex equation, that is

$A.(B+C.\overline{D})$................eqn 201

DE MORGAN'S THEOREM

For any equation, it is known in the prior art that the can be translated into an SQL instruction which, by inspection, could be degenerated into a nested set of SQLs using AND's, OR's and NOT operators, involving nested SQL. It is a general technique but it has the problem that its performance is relatively slow. On a multimillion row table, it could take hours to execute an instruction, such as a search instruction. So the problem is how do you take a general expression and make it run really fast.

The present invention has realised that addressing this problem in part involves seeing that a nesting can be removed by expansion to a set of min terms, the set of min terms then being converted into a flatten query involving zero or more join(s). In this way, a relatively long expression can be converted into a number of smaller expressions, each of which can be flattened. The flattened and smaller expressions runs much faster, thus improving performance. Also, the database could choose the best technique of evaluating the flattened terms to further enhance performance.

The present invention utilises the application of De Morgan's theorem to the service query, as an example of a way in which to simply an expression. So for the purposes of illustration, we will apply this equation to eqn 201 above. In applying De Morgan's theorem we produce what we call a sum of min terms. In this example the min terms will be:

$\Sigma_{\text{min terms}} = A.B + A.C.\overline{D}$........ .....eqn 202

What we are seeking to achieve by this step, is to represent the service query or relatively complex equation as an OR of ANDs. This is what we can call a sum of min terms, in accordance with De Morgan's theorem.

REMOVE OR's

Another aspect of the present invention is to effectively remove the 'OR's. That is an OR can be removed in a sum of minterms, by simply doing each of the minterm as a separate SQL. By applying De Morgan's theorem, OR's have been what is call pushed to the top and the AND's have been pushed to the bottom. In other words, in a relatively complex expression a very inner nested OR gets multiplied out or expanded, so that it 'pops' to the top. The invention then seeks to remove the OR's by taking each minterm and evaluating it separately. In this

REMOVE 'NOT's

If a minterm with a NOT is to be flattened, an outer join or a subselect is used as discussed above, and this is considered to slow performance.

One solution to the 'not' problem is to realise that $A\overline{B}$ can be expressed as A.(1-B), which can be written A – AB. This expression no longer includes NOT's.

Furthermore, if the service query or equation being evaluated is $A.(B+C.\overline{D})$..................................eqn 203,

The invention applies principles of Boolean logic to further expand the equation or min terms, so as to express equation 203 in a further expanded form:

A.B + (A.C – A.C.D)...........................eqn 204

Which no longer includes 'Not's.

However, a database that supports SQL may not supply a subtraction operator. Thus there is a problem in resolving the min terms as compiled using the steps noted above. In order to evaluate a subtraction, say A not B, or as expressed above A – AB, we can resolve the subtraction by evaluating A and B, the results forming a set of EIDs , then putting the results in a notEID's set, then evaluate A, check to see whether the items being returned from the A are in the notEID set ,and if they are not in the notEID set, they are added to the EID set.

So, to evaluate a subtraction, in effect, we work out all the things we don't want to keep and then work out all the things we want to keep and keep them provided they are not in the things we don't want to keep.

Thus the present invention resolves the results of the min terms as follows:

In the case of subtracted elements of the min term, the results are saved in a 'not' list, such as 'NOT EID',

In the case of positive elements of the min term, the results are saved in a list, such as 'EID', provided they are not in the 'not' list.

In the case of duplicated results, that is results already listed, do not again list, that is, ignore or disregard them.

HIGHER ORDER SUBSTRACTIONS

In the case of an equation

$A.\overline{B}.\overline{C}$.......................................eqn 205, This can be further expanded to:

A ( 1 – B ). ( 1 – C )............................eqn 206,

Eqn 206 can be even further expanded to:

A.( 1 – C - B + B.C)....................eqn 207

A – A.C – A.B + A.B.C ...................eqn 208

From here, the logic of the present invention does not strictly follow standard mathematical principals. The reason is that in the process of ignoring duplicate items, it becomes unnecessary to deal with mathematical anomalies that do try to reconcile duplicates.

Thus with eqn 208, we evaluate as follows:

results of AC are added to the NOT list

results of AB are added to the NOT list, and

results of A are added to the EID list, if they are not in the NOT list, then

Because the invention throws away duplicates, there is no need to evaluate the term A.B.C. With reference to mathematical principals, the subtraction of A.B and A.C actually subtracts A.B.C. twice. In applying present invention duplicates are not listed.

Looking at Figure 2, in a mathematical sense, if we are evaluating $A.\overline{B}.\overline{C}$, we can evaluate all of A, then subtract AC, then subtract AB. But this means graphically we have subtracted ABC twice (once in the AB subtraction and once in the AC subtraction). This is why mathematically ABC is again added in. In the present invention, however, in executing service queries, results are listed, and it is realised that AC and AB have an overlapping portion called ABC. This overlap is considered a duplicate because the results found by ABC have already been listed in AC and AB and thus they do not need to be again listed. The results of ABC are therefore not listed, and thrown away as it were. Nor does the term ABC need to be evaluated as it will not be listed (added back in as was the case in the mathematical explanation).Each of these elements, A, AC, AB, some of which include ANDs can be evaluated separately and can be flattened out, without the need for subselects.

Note that $A.\overline{B}.\overline{C}$ is an order 3 term but this can be evaluated by A-AC-AB, which contains no greater than order 2 terms. Thus the implementation using subtraction lists is considered very efficient and good performance.

It is clear that the present invention does have a very general application to the evaluation / execution of service queries. The present invention should not be limited to the examples disclosed herein, the equations being used only for illustrative purposes. Clearly the present invention can be used in respect of many different service queries, whether or not represented by equations of different simplicity or complexity.

THE CLAIMS DEFINING THE PRESENT INVENTION ARE AS FOLLOWS:

1. A method of operating a database in order to execute a service query, the method including the steps of:

receiving a service query,

applying De Morgon's theorem to the service query to obtain a sum of min terms,

evaluate each min term as a separate SQL instruction,

execute each separate SQL instruction.

2. A method as claimed in claim 1, further including the step of additionally expanding min terms to remove 'NOT's.

3. A method as claimed in claim 2, wherein the terms are expanded by application of Bolean logic.

4. A method of operating a database in order to execute a service query, the method including the steps of:

determining a SQL instruction representative of a function,

listing the results of a subtracted or negative SQL instructions in a 'NOT EID' list, listing the results of a non-subtracted or additive SQL instruction in an 'EID' list, and

not listing or ignoring results which are duplicates of listed results.

5. A method a claimed in claim 1, 2 or 3, in combination with the method of claim 4.

6. A method as claimed in any one of claims 1 to 5, in which the service query is X.500 or LDAP.

7. Method as claimed in any one of claims 1 to 6, in which the service query is a search service query.

8. Apparatus adapted to implement the method as claimed in any one of claims 1 to 7.

9. A database operable to implement the method as claimed in any one of claims 1 to 7.

10. A database as claimed in claim 9, operable to perform X.500 or LDAP services.

11. A method or device as herein disclosed.

DATED this 26th day of November, 1999

**OPENDIRECTORY PTY LTD**

WATERMARK PATENT & TRADE MARK ATTORNEYS
290 BURWOOD ROAD
HAWTHORN VIC 3122
AUSTRALIA

RCS/SH DOC 28 AU003238.doc

Fig 1.

Filter ( & (cn = Rick ) ( sn = Harvey )) ← 101

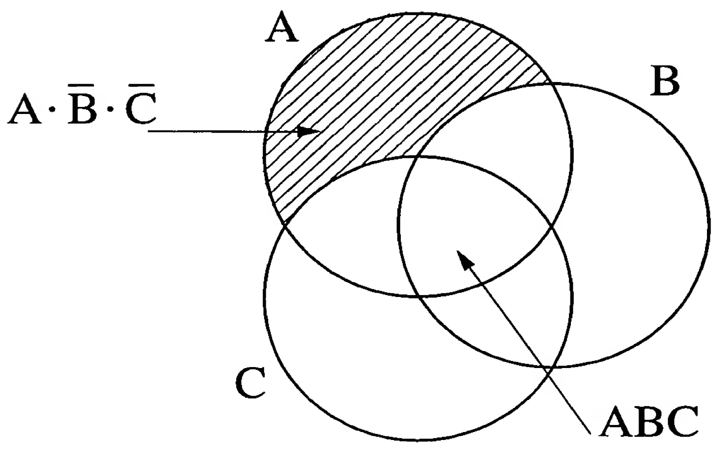
102 →

| AID | Search norm | EID |
|---|---|---|
| 3 | John | 200 |
| 3 | Rick | 101 |
| 4 | Harvey | 101 |
| 4 | Smith | 200 |

| AID | Attr |
|---|---|
| 3 | cn |
| 4 | sn |

← 103

Fig 2.


A
B
C
$A \cdot \overline{B} \cdot \overline{C}$
ABC